Comput Visual Sci (2005) 8(2): 107–118
DOI 10.1007/s00791-005-0156-5

Computing and Visualization in Science

REGULAR ARTICLE

M. Pernice · R. D. Hornung

Newton-Krylov-FAC methods for problems discretized on locally refined grids

Received: 31 January 2003 / Accepted: 30 October 2004 / Published online: 23 September 2005

Communicated by: G. Wittum

Abstract Many problems in computational science and engineering are nonlinear and time-dependent. The solutions to these problems may include spatially localized features, such as boundary layers or sharp fronts, that require very fine grids to resolve. In many cases, it is impractical or prohibitively expensive to resolve these features with a globally fine grid, especially in three dimensions. Adaptive mesh refinement (AMR) is a dynamic gridding approach that employs a fine grid only where necessary to resolve such features. Numerous AMR codes exist for solving hyperbolic problems with explicit time stepping and some classes of linear elliptic problems. Researchers have paid much less attention to the development of AMR algorithms for the implicit solution of systems of nonlinear equations.

Recent efforts encompassing a variety of applications demonstrate that Newton-Krylov methods are effective when combined with multigrid preconditioners. This suggests that hierarchical methods, such as the Fast Adaptive Composite grid (FAC) method of McCormick and Thomas, can provide effective preconditioning for problems discretized on locally refined grids. In this paper, we address algorithm and implementation issues for the use of Newton-Krylov-FAC methods on structured AMR grids. In our software infrastructure, we combine nonlinear solvers from KINSOL and PETSc with the SAMRAI AMR library, and include capabilities for implicit time stepping. We have obtained convergence rates independent of the number of grid refinement levels for simple, nonlinear, Poisson-like, problems. Additional efforts to employ this infrastructure in new applications are underway.

M. Pernice (✉)
Computer and Computational Sciences Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA
E-mail: pernice@lanl.gov

R. D. Hornung
Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551, USA
E-mail: hornung@llnl.gov

1 Introduction

Adaptive mesh refinement is a powerful technique for increasing local spatial and temporal resolution in numerical simulations of scientific and engineering problems. Structured adaptive mesh refinement (SAMR) algorithms for increasing resolution in shock problems described by hyperbolic systems of partial differential equations were pioneered in [7, 8]. Subsequent developments extended SAMR to other application areas, including incompressible flow [1, 23, 34]; flow in porous media [30]; solid mechanics [25, 41]; magnetohydrodynamics [4, 12, 22]; and laser-plasma interactions [16].

Systems of time-dependent partial differential equations are often solved using fully explicit or semi-implicit algorithms based on operator-splitting and/or time lagging. In these situations, it is typical that a linear elliptic or parabolic equation must be solved at each time integration step. Although the cost of solving these subproblems is not usually prohibitive, stability constraints limit time step size and overall accuracy can be compromised due to propagation of operator splitting errors (see, for example, [31, 37]). In contrast, fully implicit approaches are free from splitting errors, and time steps are constrained solely by accuracy considerations. This advantage is offset by the need to solve a large-scale system of nonlinear equations during each integration time step. The computational efficiency of a numerical method measures the number of numerical operations required to achieve a prescribed numerical accuracy. Thus, the relative efficiency of fully implicit methods compared to other approaches is determined by the trade-off between a reduction in the number of time steps and the larger cost of a single time step. In the end, robust and efficient methods for solving systems of nonlinear equations are needed for implicit approaches to be competitive with other solution strategies.

Inexact Newton methods [9, 14, 17] are effective for many scientific computing problems [32]. The efficiency of an inexact Newton method is largely determined by the solver used to approximate the solution of the system of

linear equations arising at each Newton iteration. An important subclass of inexact Newton methods employs Krylov subspace methods [21] to solve these linear equations. While a significant advantage of a Newton-Krylov method is that it does not require explicit formation of a Jacobian matrix, the choice of a linear preconditioner determines its efficiency. Multigrid methods are particularly effective preconditioners in the absence of local mesh refinement [32]. Fortunately, multigrid techniques can be extended to locally-refined grids found in SAMR because of the hierarchical nature of those grids. In particular, Fast Adaptive Composite grid (FAC) methods [35] have the potential to form the basis of hierarchical preconditioners for a broad class of multi-physics problems.

In this paper, we discuss an algorithm and software infrastructure that we have developed to explore the potential of Newton-Krylov-FAC methods on SAMR grids. It combines local mesh refinement capabilities with implicit time stepping procedures and inexact Newton methods. It is built largely from existing software libraries. The KINSOL [40] and PETSc [2, 3] libraries provide inexact Newton methods. The SAMRAI library [29] provides parallel SAMR capabilities and software support for the interaction between nonlinear solution methods, preconditioners, and numerical discretizations for partial differential equations.

We have addressed a number of important software and algorithm interoperability issues in the construction of interfaces to connect these pre-existing software libraries. The libraries work together efficiently and yet they are sufficiently decoupled to allow parts of the solution strategy to be replaced or customized to suit various application needs. A particularly important point that we discuss is that each phase of the Newton-Krylov-FAC solution process operates on data structures well-suited to its needs with minimal data copying or transformation. Although parallelism is not our focus here, we note that the full parallel capabilities of SAMRAI [42] are available to applications that employ this solver infrastructure. In the remainder of this paper, we focus our discussion on details of algorithmic and software coordination.

We note that our approach to providing implicit solver capabilities on locally refined grids is fundamentally different from that used in other AMR solver packages, such as UG [6], ALBERT [39] or PLTMG [5]. While these packages include sophisticated multilevel solvers (along with extensive support for grid generation, discretization, error estimation, and parallelism), the implementations are specific to the data structures employed in each respective package, and cannot be used in other contexts. In contrast, our approach leverages existing solver capabilities to the greatest extent possible, and localizes the use of grid-specific features to precisely those solver components that require these features. Besides the benefit of software reuse, this approach creates the possibility of accessing other algorithmic capabilities (such as eigensolvers, sensitivity analysis, continuation methods, and pde-constrained optimzation) that interoperate with the solver components we employ.

Aspects of structured local mesh refinement and capabilities of SAMRAI are discussed in Sect. 2. Algorithmic components are discussed in Sect. 3. This is followed by a description of the software infrastructure in Sect. 4. A demonstration calculation is presented in Sect. 5, and concluding remarks are made in Sect. 6.

2 Structured local mesh refinement

This section describes structured local mesh refinement and introduces notation that is used in subsequent sections. The notation is loosely based on that found in [35]. Let $\underline{h_k} = \{h_0, h_1, \ldots, h_k\}, k = 0, \ldots, L-1$, denote collections of mesh spacings, where h_k denotes the mesh spacing of level k and $h_{k+1} \leq h_k$. In particular, h_0 is the coarsest mesh spacing and h_{L-1} is the finest mesh spacing. An SAMR grid $\Omega^{\underline{h}_{L-1}}$ may be represented as a nested hierarchy of L grid levels $\Omega^{h_0} \supset \Omega^{h_1} \supset \cdots \supset \Omega^{h_{L-1}}$, where the coarsest grid Ω^{h_0} covers the entire computational domain Ω. In what follows, we will sometimes drop the subscript $L-1$ when referring to the entire grid hierarchy $\Omega^{\underline{h}} \equiv \Omega^{\underline{h}_{L-1}}$, and use the notation $\Omega^{\underline{h}_k}$ when referring to a subset of the grid hierarchy consisting of levels $0, \ldots, k$. Each level Ω^{h_k} consists of a union of logically rectangular regions, or patches, at the same mesh resolution h_k. This hierarchical representation facilitates implementation of operations on the composite grid $\Omega^{\underline{h}}$ by decomposing them into operations on individual levels Ω^{h_k}, which in turn are further decomposed into operations on individual patches. This property enables reuse of existing software written for regular grids, such as multigrid solvers and other numerical routines. The use of rectangular regions also allows the use of accurate discretization schemes; irregularities in the discretization are localized to well-defined regions where the resolution changes. Figure 1 shows an example grid hierarchy with $L = 3$ and two patches on each of the two finer levels. Note that refinement levels are nested, but that patches at different levels need not be nested.

Typically, we define the solution on the grid hierarchy $\Omega^{\underline{h}}$ only at spatial locations that correspond to the finest grid in the region. In particular, all of $\Omega^{h_{L-1}}$ represents part of the solution, as does the subregion of each level $\Omega^{h_{k-1}}$ that is not covered by Ω^{h_k}, for $k = 1, \ldots, L-1$. Another way to say this is that the solution is defined only in grid cells that have not been refined. Data in cells that have been refined is used to construct the solution in underlying finer cells. For instance, these cells can be used to accelerate the convergence of iterative solution procedures in a manner similar in spirit to multigrid methods. Usually the values in refined cells are defined by appropriately averaging values on the next finer grid.

SAMRAI is an object-oriented C++ library that provides a flexible and robust toolbox of classes that simplify the construction of algorithms and data management in parallel SAMR applications. The library is organized into a collection of software packages each of which contains classes

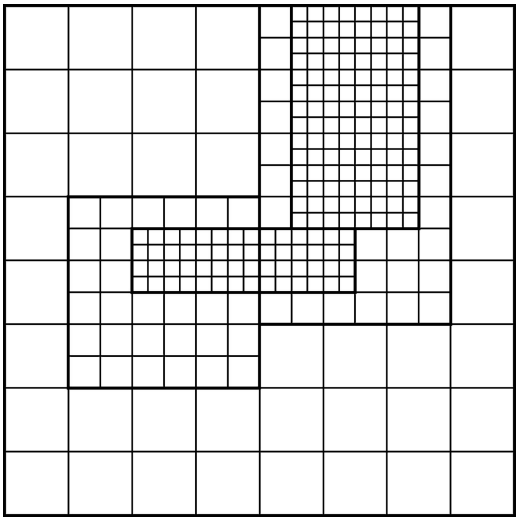

Fig. 1 Example SAMR hierarchy with three grid levels. Each finer level has two patches whose boundaries are shown with bold lines

with related functionality in SAMR grid computations. A overview of SAMRAI software organization and capabilities that facilitate application development may be found in [29]. The parallel performance and scaling properties of the adaptive capabilities of SAMRAI are discussed in [42]. Here, we briefly mention features of some packages that are most germane to this paper.

The `Hierarchy` package provides abstract index space and box calculus utilities on which most other operations on a hierarchical SAMR grid depend. Structural classes like `Patch`, `PatchLevel`, and `PatchHierarchy`, and base classes for managing variables and data on the SAMR grid hierarchy also reside here.

The `Transfer` package provides tools for interlevel and intralevel data transfers. These include base classes for operators that refine and coarsen data spatially, interpolate data in time, and fill physical boundary conditions. The package also contains communication algorithms and data transaction schedules for moving data associated with collections of variables among patches in the grid hierarchy. Such operations include filling patch data ghost cell values with data copied from neighboring patches at the same level or interpolated from patches at coarser levels. At physical domain boundaries, ghost cells are set to values appropriate for the boundary conditions and numerical methods being used.

The `Mesh` package provides capabilities for constructing an SAMR grid hierarchy and dynamically reconfiguring the hierarchy during a computation. This functionality includes interfaces to user routines for selecting grid cells for refinement. The library clusters these selected cells into box regions and constructs each new patch level, including load balancing based on either spatially-uniform or nonuniform workload estimates, according to user-supplied parameters [42].

Finally, the `Solver` package houses basic support for linear and nonlinear solvers on SAMR grids. For example, it provides interfaces to solver libraries, like PETSc [2, 3], and the KINSOL [40] and CVODE [28] solvers contained in the SUNDIALS package [27]. Also, SAMRAI provides vector classes that allow one to treat a collection of variable quantities on a subset of grid levels in an SAMR patch hierarchy as a single algebraic vector. For instance, one can define the solution vector for a Newton-Krylov nonlinear solver to contain cell-centered unknowns, node-centered unknowns, etc. Then, these quantities can be processed as a single vector entity in vector kernel operations, such as norms, inner products, and other algebraic manipulations. The solver library interfaces allow the chosen solver to manipulate SAMRAI vector data *directly* during their algorithmic execution. That is, the data is not copied from the SAMR patch hierarchy representation, which facilitates stencil-based computations, to a form on which the solver operates. The interfaces also resolve linkage issues between `C` and `C++` and simplify the implementation of user-defined routines, such as residual computations and preconditioner operations, by providing a straightforward `C++` inheritance mechanism to supply function pointers to the solver library. Much of the development of the SAMRAI `Solver` package is the direct result of work described in this paper. We will discuss these details further in Sect. 4.

3 Algorithms

A software infrastructure for solving nonlinear multi-physics problems must provide robust and efficient solver capabilities. Simultaneously, it must allow solution algorithm components to be customized for specific application needs. For treating time-dependent problems, the software must manage the time-dependent data and accommodate different time integration strategies. This section provides an overview of the necessary algorithmic ingredients.

3.1 Implicit methods for initial value problems

Many multi-physics problems can be expressed in the form of a nonlinear initial value problem (IVP)

$$\frac{\partial u}{\partial t} = f(u), \qquad u(0, \mathbf{x}) = u_0(\mathbf{x}), \quad \mathbf{x} \in \Omega \subset \mathbb{R}^d \tag{3.1}$$

together with boundary conditions specified on $\partial\Omega$. Here, f is a nonlinear function that involves spatial derivatives of its argument, u_0 is the initial condition, and d is the spatial dimension of the problem. One way to solve (3.1) is to use the *method of lines*, in which the derivatives in f are replaced by suitable discretizations. This semi-discrete approach transforms the IVP to a large-scale system of ordinary differential equations, which may be solved by a variety of methods. Solution methods are either *explicit*, where advancing the solution in time uses only past information, or *implicit*, where the advanced-time solution is determined by solving a system of equations involving unknowns at the new solution time. The method of lines leads to problems that are *stiff* so that explicit time steps must be very small to

maintain stability. However, various implicit methods suitable for stiff problems are known. In these methods, time steps are determined solely from considerations of accuracy, not stability.

Perhaps the simplest and best known example of an implicit method that also works for stiff problems is the backward Euler (BE) method. BE replaces u_t in the semi-discrete equations by a backward difference in time, and the solution at time t_{n+1} is found by solving

$$F(u) \equiv u - u^{(n)} - \Delta t f(u) = 0 \tag{3.2}$$

where $u^{(n)}$ is the solution at time t_n and $\Delta t = t_{n+1} - t_n$ is the time step. In particular, F is nonlinear when f is nonlinear. Many other methods suitable for stiff problems, such as schemes based on backward differentiation formulas (BDF) and implicit Runge-Kutta methods [26], lead to nonlinear problems that have a similar structure.

For stiff problems, the ability to use larger time steps gives implicit methods an advantage over explicit methods only if the cost of solving systems of equations such as (3.2) is not too large. This difficulty has traditionally motivated researchers to pursue alternatives to fully implicit methods, such as operator splitting and linearization. These approaches are usually problem-specific, can re-introduce stability-related restrictions on time steps, and often reduce the order of accuracy of the time discretization method. In the context of local mesh refinement, this risk is compounded by the danger of precluding any improvement in spatial accuracy resulting from increased local mesh resolution. Although these issues are still subjects of intensive efforts, recent algorithmic research has shown that the cost of solving systems such as (3.2) using inexact Newton methods can make fully implicit methods competitive in both speed and accuracy [11, 33, 36].

3.2 Inexact Newton methods

Consider a system of nonlinear equations

$$F(u) = 0. \tag{3.3}$$

A basic Newton method for solving (3.3) is given by

ALGORITHM 3.1 NEWTON'S METHOD

CHOOSE AN INITIAL APPROXIMATION u_0
  AND A TOLERANCE $\epsilon > 0$.
SET $k = 0$.
DO {
  SOLVE $F'(u_k)s_k = -F(u_k)$.
  $u_{k+1} = u_k + s_k$
  $k = k + 1$
} (WHILE $\|F(u_k)\| > \epsilon$)

At each iteration, the Newton step s_k is the solution of a system of linear equations in which the coefficient matrix is the Jacobian $F'(u_k)$ of the nonlinear system evaluated at the current approximate solution u_k. Newton's method in this form is impractical for large-scale multi-physics problems for several reasons. Storage requirements for the Jacobian F' may be high, and it may be difficult even to compute F'. Also, direct solution of the Newton equations may be too costly. Fortunately, under mild assumptions [14], Newton's method will still converge when the Newton step satisfies the inexact Newton condition

$$\|F(u_k) + F'(u_k)s_k\| \leq \eta \|F(u_k)\|, \tag{3.4}$$

where the *forcing term* η satisfies $\eta \in (0, 1)$ and may be chosen either statically or dynamically; in particular, superlinear or quadratic convergence can be recovered by dynamically choosing $\eta = \eta_k \to 0$ [18]. Consequently, s_k can be determined using an iterative method, which can dramatically reduce the cost of solving the Newton equations for large-scale problems. This leads to the following basic *inexact Newton method*:

ALGORITHM 3.2 INEXACT NEWTON'S METHOD

CHOOSE AN INITIAL APPROXIMATION u_0,
  AN INITIAL FORCING TERM $\eta \in (0, 1)$,
  AND A TOLERANCE $\epsilon > 0$.
SET $k = 0$.
DO {
  FIND s_k SATISFYING (3.4).
  $u_{k+1} = u_k + s_k$
  $k = k + 1$
  (OPTIONALLY) CHOOSE A NEW $\eta \in (0, 1)$.
} (WHILE $\|F(u_k)\| > \epsilon$)

Remark 3.1. As with classical Newton's method, inexact Newton methods are guaranteed to converge only if the initial approximation is sufficiently close to the solution. However, globalization strategies that improve the likelihood of convergence from initial approximations far from a solution are readily incorporated into Algorithm 3.2 [17].

An important subclass of inexact Newton methods is obtained when the Newton equations are approximately solved with a transpose-free Krylov subspace method [21]. These so-called *Newton-Krylov* (NK) methods possess an important property: since the transpose-free Krylov subspace method only requires Jacobian-vector products $F'(u_k)v$, the Jacobian need never be formed nor stored. Instead, these products can be approximated by finite differences

$$F'(u_k)v \approx \frac{F(u_k + \varepsilon v) - F(u_k)}{\varepsilon} \tag{3.5}$$

for a suitable choice of the differencing parameter ε. Besides saving memory, this simplifies implementation, particularly in the case where the nonlinear function F arises from discretization of a nonlinear problem on a locally refined grid. *Jacobian-free Newton-Krylov methods* result from the combination of using a Krylov subspace method to approximately solve the Newton equations with finite-difference approximations (3.5). We use GMRES [38] because it is particularly effective when a Jacobian-free approach is used.

The major cost of an inexact Newton method is the determination of the inexact Newton step s_k that satisfies (3.4). In particular, the efficiency of an NK method is determined by effective preconditioning. Preconditioners that take advantage of the hierarchical nature of SAMR grids have the potential to be particularly effective.

3.3 Hierarchical preconditioning

Considerable experience suggests that using multilevel preconditioning strategies with Newton-Krylov methods leads to robust, efficient, and scalable algorithms for a wide variety of problems [32]. It is reasonable to expect that this approach can be extended to SAMR grids, provided that we properly account for the nature of the local refinement. The Fast Adaptive Composite grid (FAC) method [35] can form the basis for extending these ideas to SAMR grids. A basic description of FAC for solving $\mathcal{L}^{\underline{h}} u^{\underline{h}} = f^{\underline{h}}$ on Ω is given by

ALGORITHM 3.3 FAC METHOD

INITIALIZE: $r^{\underline{h}} = f^{\underline{h}} - \mathcal{L}^{\underline{h}} u^{\underline{h}}$

FOR $k = L-1, \ldots, 1$ {
  SET $f^{h_k} = \mathcal{I}_{\underline{h}_k}^{h_k} r^{\underline{h}_k}$.
  SOLVE/SMOOTH $\mathcal{L}^{h_k} u^{h_k} = f^{h_k}$.
  CORRECT $u^{\underline{h}_k} = u^{\underline{h}_k} + \mathcal{I}_{h_k}^{\underline{h}_k} u^{h_k}$.
  UPDATE $r^{\underline{h}_k} = f^{\underline{h}_k} - \mathcal{L}^{\underline{h}_k} u^{\underline{h}_k}$.
}
SOLVE $\mathcal{L}^{h_0} u^{h_0} = f^{h_0}$.

FOR $k = 1, \ldots, L-1$ {
  CORRECT $u^{\underline{h}_k} = u^{\underline{h}_k} + \mathcal{I}_{\underline{h}_{k-1}}^{\underline{h}_k} u^{\underline{h}_{k-1}}$.
  SET $f^{h_k} = \mathcal{I}_{\underline{h}_k}^{h_k} (f^{\underline{h}_k} - \mathcal{L}^{\underline{h}_k} u^{\underline{h}_k})$.
  SOLVE/SMOOTH $\mathcal{L}^{h_k} u^{h_k} = f^{h_k}$.
  CORRECT $u^{\underline{h}_k} = u^{\underline{h}_k} + \mathcal{I}_{h_k}^{\underline{h}_k} u^{h_k}$.
}

In Algorithm 3.3, grid functions $u^{\underline{h}}$, $r^{\underline{h}}$, and $f^{\underline{h}}$ are defined on the grid hierarchy $\Omega^{\underline{h}}$, while u^{h_k} and f^{h_k} are defined only on the level Ω^{h_k}. $\mathcal{L}^{\underline{h}}$ refers to a linear operator that has been discretized on $\Omega^{\underline{h}}$, and $\mathcal{L}^{h_k}$ refers to a restriction of $\mathcal{L}^{\underline{h}}$ to Ω^{h_k}. In particular, $\mathcal{L}^{\underline{h}}$ accounts for the change in resolution at the interfaces between coarse and fine levels, and $\mathcal{L}^{h_k}$ is equipped with suitable boundary conditions at the boundaries of Ω^{h_k}. The operators $\mathcal{I}_{\underline{h}_{k-1}}^{\underline{h}_k} : \Omega^{\underline{h}_{k-1}} \to \Omega^{\underline{h}_k}$ perform data transfers and play a role similar to prolongation in multigrid, interpolating data in $\Omega^{h_k} \cap \Omega^{h_{k-1}}$ (at h_{k-1} resolution) to Ω^{h_k}. Additional transfer operators $\mathcal{I}_{\underline{h}_k}^{h_k} : \Omega^{\underline{h}_k} \to \Omega^{h_k}$ and $\mathcal{I}_{h_k}^{\underline{h}_k} : \Omega^{h_k} \to \Omega^{\underline{h}_k}$ serve to extract a level from $\Omega^{\underline{h}_k}$ and insert a level into $\Omega^{\underline{h}_k}$, respectively. These operations are primarily copy operations, with the exception that in an implementation $\mathcal{I}_{\underline{h}_k}^{h_k}$ often includes additional operations on $\partial\Omega^{h_k}$ to fill ghost cells. See [35] for full details.

FAC shares many elements in common with multigrid methods. Consider the V-cycle specified in Algorithm 3.3. An exact solve, which could be provided by a multigrid solver, is used on Ω^{h_0}. On $\Omega^{\underline{h}_k}$, a correction is constructed on each finer level Ω^{h_k}, $k > 0$, by first computing the residual on $\Omega^{\underline{h}_k}$ and mapping it to Ω^{h_k}. A single level correction is then calculated on Ω^{h_k} by solving (or smoothing) a residual equation on Ω^{h_k}. The solution is corrected on $\Omega^{\underline{h}_k}$ and the residual is updated on $\Omega^{\underline{h}_k}$. The residual is then restricted to $\Omega^{\underline{h}_{k-1}}$. The procedure then recurses through coarser levels on the SAMR grid. Upon return from the recursion, the solution is corrected on $\Omega^{\underline{h}_k}$, followed by an update of the residual on $\Omega^{\underline{h}_k}$. A post-smoothing sweep on Ω^{h_k} and a second correction on Ω^{h_k} completes the cycle. Other cycling strategies, such as a coarse-to-fine slash cycle (suitable for problems where a good initial approximation is not known) or F-cycles (suitable for robustness when the coarsest grid is not solved exactly) are also possible.

We emphasize that FAC is driven by residuals that are calculated on $\Omega^{\underline{h}_k}$ and accounts for changes in resolution at coarse-fine interfaces via the operators $\mathcal{L}^{\underline{h}_k}$ obtained through some discretization on $\Omega^{\underline{h}_k}$. The details of the discretization depends on the underlying uniform grid discretization and the specific problem being solved.

4 Software infrastructure

The SAMRAI AMR library provides parallel data management and supports the interaction of the various algorithmic pieces in our implicit-timestepping, Newton-Krylov-FAC solution strategy. In this section, we describe the essential design features of implicit timestepping and interfaces to solution methods for systems of nonlinear equations on an SAMR grid hierarchy. Nonlinear solver capabilities, such as inexact Newton methods, are provided through interfaces to the solver libraries KINSOL and PETSc. We are primarily concerned with software design that promotes algorithmic flexibility and efficient implementation of different numerical aspects associated with solving nonlinear problems. Discussion of parallel scaling of data communication and adaptive meshing operations in SAMRAI may be found in [42].

Our approach solves key problems associated with compartmentalizing algorithmic control and treating nonuniform data structures across software libraries. The main difficulties result from choices made by solver library developers who wish to provide specific algorithmic content in a general manner for use in a broad range of applications. Typically, the data structures in these libraries, such as vectors and matrices, are designed for high performance of the target algorithms. This approach is very useful and flexible from a solver perspective. However, application developers

who use the libraries may need to provide data for their problem in a form suited to the solvers but less-suited for their numerical methods. The result can be a monolithic application code implementation requiring data copies between the solvers and the application. Also, it may be difficult for application developers to modify or employ different aspects of solver libraries.

For example, *hypre* [20] provides powerful multigrid algorithms, but requires problem data to be transformed into matrix and vector representations that are highly tuned to linear algebra needs and that do not provide capabilities for locally-refined grids. Multilevel preconditioners, such as FAC, can be expressed abstractly in terms of operations on levels in a locally-refined grid hierarchy. It is straightforward to use *hypre* operations, such as solving on a level, but other finer-grained capabilities, such as smoothing on a level, are not accessible. Also, customized operations, such as specialized prolongation and restriction, are best performed using abstractions that naturally expresses the necessary operations.

Nonlinear solver engines are typically layered atop linear solver libraries and likewise do not provide the means to manage the data complexity associated with locally-refined grids. Although KINSOL and PETSc provide their own vector types, they fortunately allow these vectors to be replaced with other suitable data structures. This is extremely valuable, since evaluation of nonlinear residuals and Jacobian-vector products on an SAMR grid hierarchy is facilitated by exploiting the multilevel representation.

In the following sections, we explore these issues further and discuss how our software approach accommodates a broad range of algorithmic generality and flexibility.

4.1 Implicit time discretization

Any practical implementation of an implicit time integration method should include several features besides setting up and solving a method-specific system of equations such as (3.2). If variable timestepping is used, some means for selecting the next timestep must be provided. In addition, solution of the implicit equations can be aided by providing an improved initial approximation to the time-advanced solution; for example, one may extrapolate in time from earlier solution values. Acceptability of the time-advanced solution delivered by the nonlinear solver must be determined also. If it is unacceptable, some action must be taken to recompute the solution, perhaps with a smaller timestep. In general, this requires problem-specific evaluation of the integrated solution. Additionally, failure of the nonlinear solver to converge to specified accuracy should be handled gracefully when possible. Finally, data structures that manage the time-dependent solution process must be updated. Following is an outline for the backward Euler method that incorporates these features.

ALGORITHM 4.1 BACKWARD EULER METHOD

SET $t = t_0$, $n = 0$, $\mathrm{u}^{(0)} = \mathrm{u}_0$.
CHOOSE AN INITIAL TIME INCREMENT Δt.
$t = t + \Delta t$.
DO {
  DO {
    CHOOSE INITIAL APPROXIMATION FOR $\mathrm{u}^{(n+1)}$.
    SOLVE (3.2) TO OBTAIN $u^{(n+1)}$.
  } (UNTIL $u^{(n+1)}$ IS SATISFACTORY)
  UPDATE SOLUTION.
  CHOOSE A NEW Δt.
  $t = t + \Delta t$, $n = n + 1$.
} (WHILE $t \leq t_{final}$)

Backward Euler implicit time discretization serves as a useful archetype for exploring the issues in more sophisticated and accurate integration methods. Currently, we provide an implementation of this method. Higher-order schemes, such as Crank-Nicolson, BDF and implicit Runge-Kutta methods, can readily be accommodated within the same framework.

Figure 2 depicts the organization and relationships among the solution components in our design. The figure shows the abstract interface classes and concrete implementations obeying those interfaces as well as most of the important member functions of each class. The `ImplicitIntegrator` object is the main driver for the integration process. It coordinates routines that implement the spatial and temporal aspects of the numerical approximation via the `NonlinearSolverStrategy` and `ImplicitIntegratorStrategy` interfaces, respectively.

The `ImplicitIntegratorStrategy` class defines the interface for methods that treat the temporal discretization. Each "strategy" class in our design follows the *Strategy* object-oriented design pattern [24]. In particular, a strategy is an abstract base class that defines the interface for a concrete implementation that is provided by a derived subclass. For example, the application code must define the variables comprising the solution vector, compute an initial guess for the next time integration step, select the timestep Δt, determine whether a solution is acceptable, and manage storage for the solution and other problem-dependent quantities when an acceptable solution is found. These operations are declared in the `ImplicitIntegratorStrategy` interface.

4.2 Interfaces to nonlinear solvers

A simple interface for solving a system of nonlinear equations is defined by the `NonlinearSolverStrategy` class. Any nonlinear solver that implements this interface can be used. Currently, we provide two concrete nonlinear solvers by using the suites of inexact Newton methods found in the KINSOL and PETSc libraries.

The `SNES_SAMRAIContext` class exposes PETSc functionality and the `KINSOL_SAMRAIContext` class

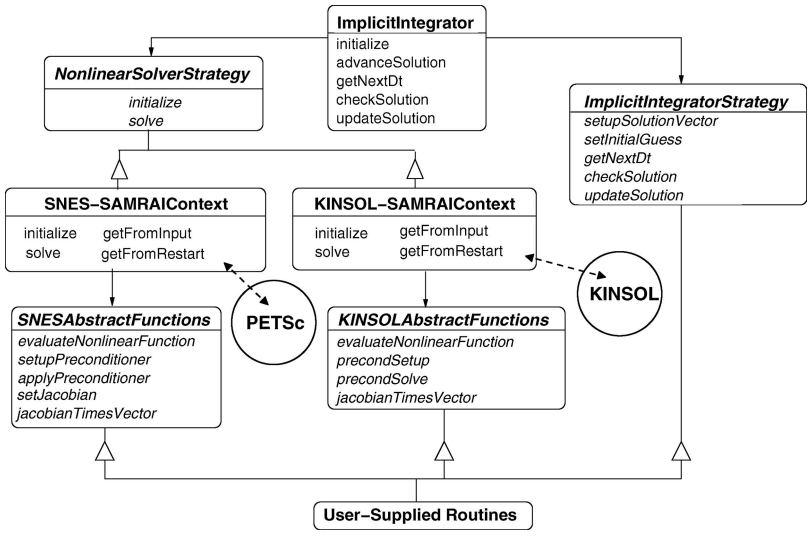

Fig. 2 Organizational structure of implicit timestepping and nonlinear solver classes and interfaces. Abstract interfaces are specified with italicized type, while concrete classes are denoted with standard type. Inheritance is indicated with an open triangle pointing toward the base class, and a solid arrow indicates when an object maintains a reference (e.g., a pointer) to another object. Most of the more important member functions for each class are also indicated. Note that the solver libraries, KINSOL or PETSc, interact directly with the "wrapper" classes, either `KINSOL-SAMRAIContext` and `SNES-SAMRAIContext`, only

does the same for KINSOL. These two classes are *wrapper*s that follow the *Adapter* object-oriented design pattern [24]. User code may either call solver library routines directly or access the solvers through these wrapper classes. However, the wrappers expose the necessary functionality of their corresponding solver libraries in a uniform fashion that makes use of either solver library in an application simple and straightforward. The wrappers also perform useful functions in our design beyond implementing the `NonlinearSolverStrategy` interface. They provide additional capabilities, such as input and restart file processing, provided by SAMRAI so that these things can be done uniformly by the application and solvers. Finally, the wrappers eliminate complexity for users by handling the details of linking `C` and `C++` code, especially when the `C` code calls `C++` class methods. Solver-specific interfaces to user code are defined in the `SNESAbstractFunctions` and `KINSOLAbstractFunctions` abstract base classes. These classes provide an inheritance-based, object-oriented mechanism for providing user routines to the solvers.

The ability to leverage the KINSOL and PETSc libraries is possible since each of their inexact Newton method implementations is written in terms of operations on vectors, as described in Sect. 3.2, and these vectors can be replaced. We replace the vectors from these libraries with SAMRAI vectors, which provide vector data management and operations for an SAMR grid hierarchy managed by SAMRAI. This situation is described in more detail below.

4.3 Sharing solution vectors among software components

Typically, application codes use vectors as containers for any number of variables that possibly have different grid centerings, such as cell-centered and node-centered quantities. In the simplest usage, a single variable defined on a single global grid is mapped to a single-indexed vector using some ordering scheme; for a structured grid, lexicographical ordering is the usual choice. When multiple variables are present in a simulation, a variety of mappings are possible. For example, storage can be mapped to single-index vector locations one variable at a time, or one grid cell at a time (sometimes referred to as block mapping). When mixed centerings are used, as happens, for example, in a staggered grid discretization for fluids in which the velocity field is face-centered and the pressure is cell-centered, more care is needed due to the different number of grid locations for each type of grid centering. Usually, the mapping of unknowns represented by multiple variable grid quantities to a vector that can be processed by a solver library is the responsibility of application code developer employing the solver. SAMRAI vectors provide a natural way to employ solvers to solve problems that involve more than one variable quantity on the SAMR grid hierarchy as we describe below.

The concept of a vector readily extends to distributed memory environments by including additional bookkeeping for mapping portions of vectors onto different processors. Often both global indices and local indices are employed. Operations can readily be defined on such distributed vectors without difficulty; in fact this can be done in a way that makes the location of data completely transparent to an application code.

Many applications maintain a dual description of their data, if only implicitly. In one description, data resides in contiguous single-indexed locations. In another, the grid-based nature of the data is used through references to nearest neighbors using offsets into other single-indexed locations. In fully unstructured calculations, this idea is generalized by

supplementing the vector representation with neighbor lists associated with each vector index.

A SAMR grid hierarchy introduces new complications. For example, all data values at all grid locations on all grid levels are not typically considered to be part of the numerical solution. Although data may be allocated on all levels in a particular region of the domain, the finest grid in a region usually contains the desired solution values. Moreover, the grid can change dynamically during the course of an adaptive calculation. This will change the dimension of the vector space for the problem as well as which grid points on each level hold valid solution values.

A `SAMRAIVector` object handles these complexities transparently for the user and the solver by translating vector routines into operations for data stored on a SAMR grid hierarchy. Each `SAMRAIVector` simply maintains a pointer to the SAMR grid on which the variables are defined and holds information for accessing the variable data on patches in the grid hierarchy, such as an integer index for the location of the data in an array of patch data objects. Vector operations suitable for different grid centerings are also provided and shared by all vector objects. Thus, a `SAMRAIVector` has negligible storage overhead and derives all of its functionality from SAMRAI grid and data management capabilities. In addition, a `SAMRAIVector` object never changes to accomodate changes in the algebraic structure of the vector due to regridding operations. Grid changes affect the patch configuration of the SAMR hierarchy only. Such flexibility is not generally found in solver libraries. In fact, changing a vector supplied by a solver library usually requires one to change the solver as well by reinitializing its state. Figure 3 depicts the structure of a `SAMRAIVector` object.

It is also possible to register a *weight* quantity that applies to one or more variable components registered with a `SAMRAIVector`. This weight can serve several purposes. For one, it can be used to mask data on coarse grid cells that are covered by finer cells. It can also be used to store quadrature information that reflects the manner in which the discretization is derived. For example, it is natural to define a norm by summing over the levels in the hierarchy, patches $\mathcal{P}$ in each level, and indexes in each patch:

$$\|f^2\| = \frac{1}{V} \sum_{k=0}^{L-1} \sum_{\mathcal{P} \in \Omega^{h_k}} \sum_{(i,j) \in \mathcal{P}} |f_{i,j}|^2 V_{i,j}$$

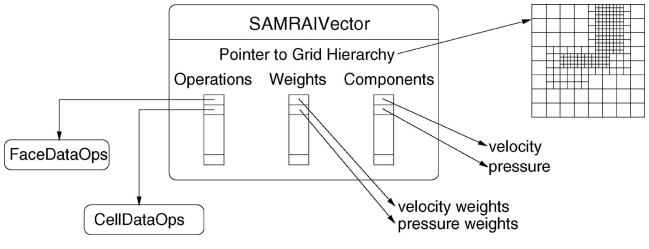

Fig. 3 Layout of a `SAMRAIVector` object. Each vector holds information for manipulating data associated with the variables registered with the vector, such as applying vector kernel operations for the appropriate data centering, and weights for treating multi-resolution data

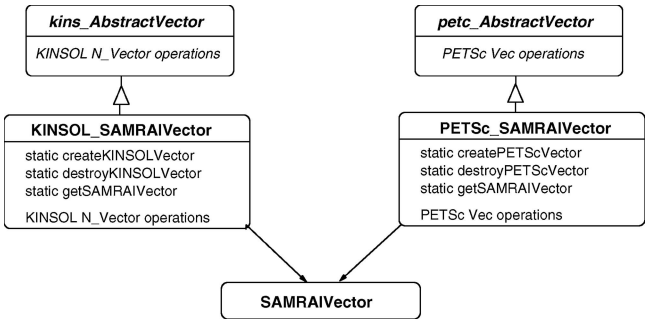

Fig. 4 A `SAMRAIVector` object is wrapped to provide operations specific to the KINSOL and PETSc solver libraries

where V is the volume of the domain Ω and

$$V_{i,j} = \begin{cases} \text{vol}(\Omega_{i,j}) & \text{if cell } (i,j) \in \Omega^{h_k} - \Omega^{h_{k+1}} \\ 0 & \text{otherwise} \end{cases}$$

This definition of a norm is the natural discrete analog of the continuous L^2 norm and has the property that $\|1\| = 1$, independent of the number of levels and grid cells in the SAMR grid configuration. This is accomplished by a suitable initialization of each weight quantity and is fairly straightforward to maintain as the grid configuration changes during dynamic regridding using SAMRAI utilities.

As noted earlier, KINSOL and PETSc provide their own vectors, called `N_vectors` and `Vecs`, respectively. These structures have distinct storage schemes and operations specific to the implementation of their respective solver engines. SAMRAI exploits the fact that these solver libraries allow their vector structures to be replaced by supplying classes that wrap `SAMRAIVector` objects as solver-specific vectors. This is illustrated in Fig. 4.

The SAMRAI library provides abstract base classes `kins_AbstractVector` and `petc_AbstractVector` to facilitate the use of `C++` vector objects in the solver libraries, which are written in `C`. These interfaces are designed to be independent of SAMRAI; their primary role is to simplify linking `C` and `C++` code. Each subclass `KINSOL_SAMRAIVector` and `PETSc_SAMRAIVector` wraps a `SAMRAIVector` for use by the corresponding solver library. Each wrapper translates calls to vector kernel operations made within the solver to routines supplied by the `SAMRAIVector` class. Typical usage of one of these wrapper classes involves first creating a `SAMRAIVector` object, then "wrapping" it via a *static* creation method which also creates a vector object recognized by either KINSOL or PETSc. The static methods facilitate creation and destruction of vectors and wrappers in application code as well from within solver library code. There is negligible storage and performance overhead associated with this decoupling, and software maintenance is simplified as each of the libraries changes independently.

One additional point related to vector storage overhead is worth noting. While ghost cells are needed for most solution variables when they are used in stencil-based operations, linear algebraic operations in solver libraries do not require

them. KINSOL and PETSc both create internal workspace by cloning vectors provided by the application. Including ghost cells in these cloned vectors could lead to a potentially unacceptable storage overhead when there is a large number of small patches in a large three-dimensional problem, or when the dimension of the Krylov subspace is large. Also, embedding ghost cells in a vector reduces the efficiency of vector operations by requiring non-unit strides between vector elements separated by ghost cells.

In our approach, the solution vector defined for the solver typically is defined to have patch data components without ghost cells. Since the solution vector is passed to the solver during initialization, all internal state vectors cloned from that reference vector will also have data with no ghost cell storage. Then, it is the responsibility of each application code routine to supplement incoming vector data with ghost cell storage and set data in the ghost regions when necessary. Also, before control is returned to the nonlinear solver, changes to vector values (excluding ghost data) must be passed back to the proper vector storage used by the solver through local (i.e., on processor) copies. With minor variations, this approach is rather standard for applications developed for distributed memory high-performance computing systems. Also, upon reflection, it should be apparent that placing these data management responsibilities on user routines is desirable for multi-physics problems involving complicated boundary conditions and sophisticated composite grid numerical discretizations. The SAMRAI `Transfer` package provides powerful capabilities to manage these issues in a general application code.

4.4 Preconditioning

The implicit timestepping classes and interfaces to the nonlinear solver libraries are designed to be general and flexible. In particular, the interfaces to the nonlinear solvers provide the full capabilities of the underlying software. However, it is much more difficult to deliver the same degree of generality and flexibility for preconditioning due to the variety of application requirements and possible algorithmic strategies. Since preconditioner performance is key to overall algorithm performance, it is important to adopt an approach that sufficiently decouples the preconditioner from the rest of the solution strategy so that the exploration of various preconditioner techniques is not precluded.

Problem-specific aspects of FAC can be found by examining Algorithm 3.3. In particular, it is apparent that a generic implementation must allow an application to fully specify the discrete operator $\mathcal{L}^{\underline{h}}$, including physical boundary conditions and discretization at changes in grid resolution. Further, multigrid experience demonstrates that defining the interlevel transfer operator $\mathcal{I}^{h_k}_{h_{k-1}}$ in terms of the grid geometry alone often leads to difficulties for certain problems, such as those involving diffusion operators with discontinuous coefficients. Such difficulties can often be alleviated by using operator-dependent interlevel transfers, which often form the basis of "black-box" multigrid packages [13, 15]. While geometry-based interlevel transfers can use capabilities that are already built into SAMRAI, operator-dependent interlevel transfers, which are defined in terms of the particular equations being solved, are outside the scope of the framework.

We implement an FAC preconditioner in two separate components. The first supplies the basic FAC scheduling algorithm for visiting levels of $\Omega^{\underline{h}}$ and is independent of the problem being solved. The second is a problem-specific component in which application developers define the problem and implement interlevel transfer operations via inheritance from an interface that follows the strategy design pattern. At this stage of development, the capabilities are sufficiently flexible to treat diffusion operators with discontinuous coefficients as well as problems having multiple variables per gridpoint. However, we consider our implementation rudimentary and will present more details in future work.

The benefits of such an approach are becoming apparent in an ongoing effort that uses this solver infrastructure to extend the methods of [11] to locally refined grids. In this work, multilevel solves for a Poisson problem, a convection-diffusion problem, and a convection-diffusion problem involving a tensor-valued diffusion coefficient are needed to implement the preconditioner. The above strategy allows us to coordinate these parts of the overall solution approach by forcing us to implement only the operator-specific aspects of each solver, re-using the component that implements the FAC scheduling algorithm.

5 Example calculations

We demonstrate the capabilities of the software and algorithmic components described on an unsteady version of the Bratu problem:

$$\frac{\partial u}{\partial t} = \Delta u + \lambda e^u + f, \quad t \geq 0, \quad (x, y) \in \Omega \equiv [0, 1]^2 \qquad (5.1)$$

with initial and boundary conditions

$$u(x, y, 0) = 0 \qquad (x, y) \in \Omega$$
$$u(x, y, t) = 0 \qquad t > 0,\ (x, y) \in \partial\Omega$$

where the source term f is determined by specifying the exact solution

$$u(x, y) = tx(1 - x)y(1 - y)\mathrm{e}^{-\frac{(x-\frac{1}{2})^2+(y-\frac{1}{2})^2}{\sigma^2}}.$$

The parameter σ provides a mechanism to adjust the size of the solution feature requiring enhanced resolution. Both the number of refinement levels and the size of the base grid are varied. Static refinement regions are defined simply as

$$\Omega^k = \left[\frac{2^k - 1}{2^{k+1}}, \frac{2^k + 1}{2^{k+1}}\right]^2, \quad k = 0, 1, 2, \ldots$$

with $\Omega^0 = \Omega$.

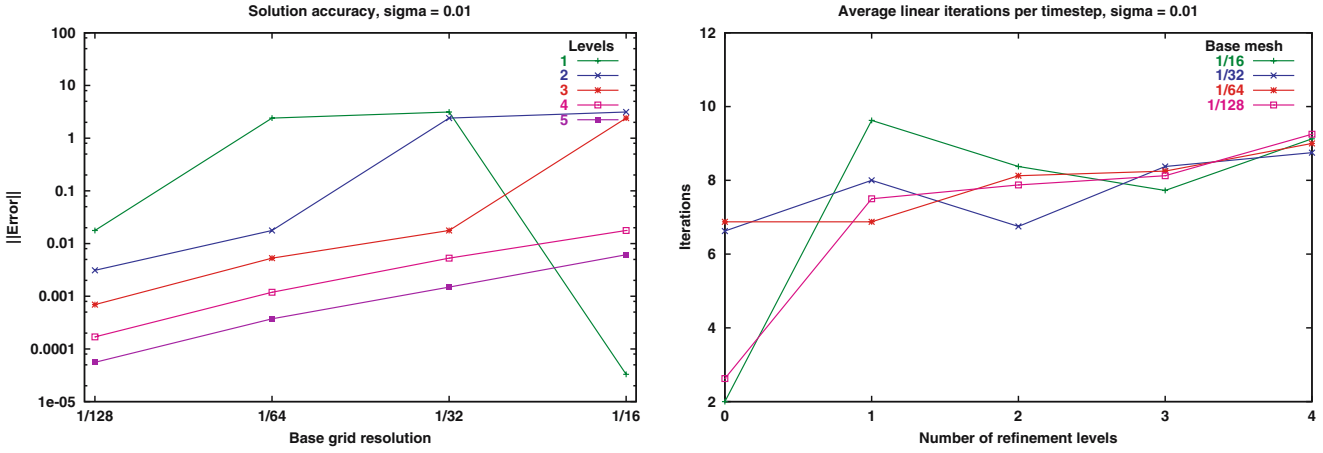

Fig. 5 Verification study of Newton-Krylov-FAC method. The top plot depicts the error, measured with a weighted L_2 norm, as a function of base mesh size, for several different degrees of local refinement. The bottom plot depicts the amount of work needed to compute the solution as a function of the number of refinement levels, for a variety of different base meshes

The concrete FAC solver used in this example uses red-black Gauß-Seidel smoothing on the finer levels. Interlevel transfers use bilinear interpolation for both restriction and prolongation. The coarsest level is solved to a prescribed tolerance of 10^{-4} using the PFMG solver from *hypre* [20].

Except for cells adjacent to changes in resolution, the problem can be discretized using the usual finite volume approach. Near coarse/fine interfaces, this must be modified in two ways. First, for cells on the coarse side of the coarse/fine interface, the coarse cell flux at the face shared by two fine cells must be set to the sum of the two fine cell fluxes. Second, the computation of the fine cell fluxes must account for the change in resolution as well as the fact that cell-centered data at different resolution is not properly aligned. The fine fluxes are computed using a simple difference of coarse and fine data together with a tangential correction that is computed on the fine side of the interface [19].

The solution is advanced to a final time of $t_{final} = 1$. Results for $\sigma = 0.01$ and time step $\Delta t = 0.125$ appear in Fig. 5. The base grid with $h = 1/16$ and no refinement fails to resolve the solution feature; three further local refinements are needed before adequate resolution is obtained. In this case the finest grid has a mesh size of $h = 1/128$ and achieves slightly better accuracy than is obtained with this global mesh size using 98% fewer gridpoints. For a given number of refinement levels, using a finer base mesh yields roughly second-order improvements in accuracy; for a given base mesh, adding refinement levels yields slightly less than second-order improvements in accuracy. Further, for each base grid size, the number of linear iterations per time step is nearly independent of the number of refinement levels.

6 Conclusions

We have discussed algorithm needs for solving large-scale nonlinear problems on locally refined grids, and have described a software infrastructure that supports these needs with a significant degree of flexibility to explore algorithmic choices. Two particularly important contributions have been described. First, effective software reuse has been achieved and nontrivial software interoperability hurdles have been resolved. Newton-Krylov nonlinear solver implementations in the PETSc and KINSOL libraries are expressed in terms of high-level operations on vectors. Interoperability between the solver libraries and SAMRAI is accomplished by redefining these operations so that the solvers can operate directly on SAMR grid hierarchy data. Furthermore, in the operations provided by the solvers, data is managed by SAMRAI, and, other than user-defined operations, no data copies need to be made. This sort of data structure decoupling also allows hierarchical preconditioning strategies using SAMR grid operations provided by SAMRAI to be easily coordinated with the nonlinear solver libraries. Note that this contrasts with the approach taken in [10], where the preferred strategy was to copy data between the solver package and the grid package. Second, algorithmic flexibility and robustness was achieved. Time integration strategies, nonlinear solver implementations, and preconditioner algorithms are decomposed and sufficiently separated from problem-specific concerns so that algorithmic variations may be explored fairly easily for a given problem. Also, we demonstrated convergence rates independent of the number of mesh refinement levels for a simple nonlinear heat equation. While this is a minimum requirement that must be met before extending these strategies to more realistic problems, current work is showing that the approach and tools described in this paper are applicable to more complex problems.

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract number W-7405-Eng-48 and by University of California Los Alamos National Laboratory under contract number W-7405-Eng-36. UCRL-JC-151614 and LA-UR 03-0619.

References

1. Almgren, A.S., Bell, J.B., Colella, P., Howell, L.H., Welcome, M.L.: A conservative adaptive projection method for variable density incompressible Navier-Stokes equations. J. Comput. Phys. **142**, 1–46 (1998)
2. Balay, S., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Tech. Rep. ANL-95/11 — Revision 2.1.3, Argonne National Laboratory, (2002) See http://www-unix.mcs.anl.gov/petsc/petsc-2/
3. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In: Arge, E., Bruaset, A.M., Langtangen, H.P. (eds.) Modern Software Tools in Scientific Computing pp. 163–202. Birkhauser Press, Cambridge, MA (1997)
4. Balsara, D.S.: Divergence-free adaptive mesh refinement for magnetohydrodynamics. J. Comput. Phys. **174**, 614–648 (2001)
5. Bank, R.E.: PLTMG: A software package for solving elliptic partial differential equations Users Guide 9.0, tech. rep., Department of Mathematics, University of California San Diego (2004)
6. Bastian, P., Birken, K., Johannsen, K., Lang, S., Neuß, N., Rentz-Reichert, H., Wieners, C.: UG–A flexible software toolbox for solving partial differential equations. Comput. Visual. Sci. **1**, 27–40 (1997)
7. Berger, M.J., Colella, P.: Local adaptive mesh refinement for shock hydrodynamics. J. Comput. Phys. **82**, 64–84 (1989)
8. Berger, M.J., Oliger, J.: Adaptive mesh refinement for hyperbolic partial differential equations. J. Comput. Phys. **53**, 484–512 (1984)
9. Brown, P.N., Saad, Y.: Hybrid Krylov methods for nonlinear systems of equations. SIAM J. Sci. Statist. Comput. **11**, 450–481 (1990)
10. Buschelman, K.R., Gropp, W.D., McInnes, L.C., Smith, B.F.: PETSc and Overture: Lessons learned developing an interface between components. In: Boisvert, R.F., Tang, P.T.P. (eds.) The Architecture of Scientific Software. Kluwer Boston (2001)
11. Chacon, L., Knoll, D.A., Finn, J.M.: An implicit, nonlinear reduced resistive MHD solver. J. Comput. Phys. **178**, 15–36 (2002)
12. de Zeeuw, D.L., Gombosi, T.L., Groth, C.P.T., Powell, K.G., Stout, Q.F.: An adaptive MHD method for global space weather simulations. IEEE Trans. Plasma Sci. **28**, 1956–1965 (2000)
13. de Zeeuw, P.M.: Matrix dependent prolongations and restrictions in a black box multigrid solver. J. Comput. Appl. Math. **3**, 1–27 (1990)
14. Dembo, R.S., Eisenstat, S.C., Steihaug, T.: Inexact Newton methods. SIAM J. Numer. Anal. **19**, 400–408 (1982)
15. Dendy, Jr., J.E.: Black box multigrid. J. Comput. Phys. **48**, 366–386 (1982)
16. Dorr, M.R., Garaizar, F.X., Hittinger, J.A.F.: Simulation of laser plasma filamentation using adaptive mesh refinement. J. Comput. Phys. **177**, 233–263 (2002)
17. Eisenstat, S.C., Walker, H.F.: Globally convergent inexact Newton methods. SIAM J. Optimization **4**, 393–422 (1994)
18. Eisenstat, S.C., Walker, H.F.: Choosing the forcing terms in an inexact Newton method. SIAM J. Sci. Comput. **17**, 16–32 (1996)
19. Ewing, R.E., Lazarov, R.D., Vassilevski, P.S.: Local refinement techniques for elliptic problems on cell–centered grids. I: Error analysis. Math. Comp. **56**, 437–461 (1991)
20. Falgout, R.D., Yang, U.M.: Hypre a library of high performance preconditioners. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J.J., Hoekstra, A.G. (eds.) Computational Science - CARS 2002 Part III, vol. 2331 of Lecture Notes in Computer Science pp. 632–641. New York. Springer-Verlag (2002) See http://www.llnl.gov/CASC/hypre/
21. Freund, R.W., Golub, G.H., Nachtigal, N.M.: Iterative Solution of Linear Systems. Acta Numerica, pp. 87–100 (1992)
22. Friedel, H., Grauer, R., Marliani, C.: Adaptive mesh refinement for singular current sheets in incompressible magnetohydrodynamics flow. J. Comput. Phys. **134**, 190–198 (1997)
23. Fuchs, L.: A local mesh refinement technique for incompressible flows. Computers and Fluids **14**, 69–81 (1986)
24. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA (1995)
25. Garaizar, X.F., Trangenstein, J.: Adaptive mesh refinement and front-tracking for shear bands in an antiplane shear model. SIAM J. Sci. Comput. **20**, 750–779 (1998)
26. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. vol. 14 of Springer Series in Computational Mathematics. Springer-Verlag, Berlin (1991)
27. Hindmarsh, A.C.: SUNDIALS SUite of Nonlinear and DIfferential/ALgebraic equation Solvers. See http://www.llnl.gov/CASC/sundials/ (in press)
28. Hindmarsh, A.C., Taylor, A.G.: PVODE and KINSOL: Parallel software for differential and nonlinear systems. Tech. Rep. UCRL–ID–129739, Lawrence Livermore National Laboratory (1998)
29. Hornung, R.D., Kohn, S.: Managing application complexity in the SAMRAI object-oriented framework. Concurrency Comput.: Pract. Exp. **14**, 347–368 (2002). See http://www.llnl.gov/CASC/SAMRAI/
30. Hornung, R.D., Trangenstein, J.A.: Adaptive mesh refinement and multilevel iteration for flow in porous media. J. Comput. Phys. **136**, 522–545 (1997)
31. Knoll, D.A., Chacón, L., Margolin, L.G., Mousseau, V.A.: On balanced approximations for time integration of multiple time scale systems. J. Comput. Phys. **185**, 583–611 (2003)
32. Knoll, D.A., Keyes, D.E.: Jacobian-free Newton-Krylov methods: a survey of approaches and applications. J. Comput. Phys. **193**, 357–397 (2004)
33. Knoll, D.A., Rider, W.J., Olson, G.L.: Nonlinear convergence, accuracy, and time step control in nonequilibrium radiation diffusion. J. Quant. Spectrosc. Ra. **70**, 25–36 (2001)
34. Martin, D.F., Colella, P.: A cell-centered adaptive projection method for the incompressible Euler equations. J. Comput. Phys. **163**, 271–312 (2000)
35. McCormick, S.F.: Multilevel Adpative Methods for Partial Differential Equations. SIAM, Philadelphia, PA (1989)
36. Mousseau, V.A., Knoll, D.A., Reisner, J.M.: An implicit nonlinearly consistent method for the two-dimensional shallow-water equation with Coriolis force. Mon. Weather Rev. **130**, 2611–2625 (2002)
37. Ober, C.C., Shadid, J.N.: Studies on the accuracy of time integration methods for the radiation diffusion equations. J. Comput. Phys. **195**, 743–772 (2004)

38. Saad, Y., Schultz, M.H.: GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Statist. Comput. **7**, 856–869 (1986)
39. Schmidt, A., Siebert, K.G.: ALBERT – software for scientific computations and applications. Acta Math. Univ. Comenianae **70**, 105–122 (2001)
40. Taylor, A.G., Hindmarsh, A.C.: User documentation for KINSOL, a nonlinear solver for sequential and parallel computers. Tech. Rep. UCRL–ID–131185, Lawrence Livermore National Laboratory (1998)
41. Trangenstein, J.A.: Adaptive mesh refinement for wave propagation in nonlinear solids. SIAM J. Sci. Comput. **82**, 94–121 (1989)
42. Wissink, A., Hysom, D., Hornung, R.: Enhancing scalability of parallel structured AMR calculations. In: Proceedings of the 17th ACM International Conference on Supercomputing (ICS03), pp. 336–347. San Francisco, CA (2003)